AAAI 2004 Spring Symposium Series (Submitted October 2003) *ZANG-S4*

Split Knowledge Level Modeling

Michael A Zang
Senior Software Engineer
CDM Technologies, Inc
2975 McMillan Avenue, suite 272
San Luis Obispo, CA, 93401
mzang@cdmtech.com

Abstract

Traditional approaches to building intelligent information systems employ an object model to define a representational structure for the information of interest within the target domain of the system. At runtime, the model provides a constrained template for the creation of the individual object instances and relationships that together define the state of the system at a given point in time. The ontology also provides a vocabulary for expressing domain knowledge typically in the form of rules (declarative knowledge) or methods (procedural knowledge). Agents operating within the system utilize the encoded knowledge to progress the state of the system towards the specific goals indicated by the users. While this approach has been very successful, it has some drawbacks, particularly in regards to the development of agent based decision support systems. Regardless of the implementation paradigm the knowledge applied by the agents is essentially buried in the code and therefore inaccessible to most domain experts. The knowledge also tends to be very domain specific and is not extensible at runtime. This paper describes the use of an explicit knowledge level within the ontology to mitigate the identified drawbacks while reducing both the number of classes and rules required.

Introduction

This paper employs a simple example to demonstrate the benefits of explicit knowledge level modeling. Knowledge level models are employed in most of the software projects currently being developed at CDM Technologies, Inc, which specializes in the development of collaborative decision support systems for large government and private organizations, particularly in the field of maritime logistics. A simple medical diagnostic model and accompanying agent rules capable of diagnosing infection types and of recommending actions to assist in the diagnosis is used for the example. A traditional model and the corresponding rules are first described, then a corresponding knowledge level model is described and the two are compared along the way. Finally, summarizing conclusions are provided, which identify the strengths and weaknesses associated with employing an explicit knowledge level and guidance provided as to when it should be considered for use.

The ideas described by this paper are based on the work of Martin Fowler described in his book Analysis Patterns, Reusable Object Models (Fowler 1997a) and the work of David Hay described in his book Data Model Patterns, Conventions of Thought (Hay 1996). This paper assumes but does not require a rudimentary knowledge of the basic concepts of object-oriented modeling. A good introduction to this subject can be found in the book Inside the Object Model (Papurt 1995). All the figures in this paper use a small subset of the notations defined by the Unified Modeling Language (UML). A concise summary of UML can be found in UML Distilled by Martin Fowler (Fowler 1997b).

A Traditional Domain Model

Traditionally, a domain model utilizes a statically compiled ontology that virtually mirrors the real-world entities associated with the targeted system domain. Model development is followed by developing agent rule sets, which are grounded in the structured vocabulary the model provides, to produce the requisite behavior for each individual agent. The remainder of this section describes a simple ontology for the medical diagnostic domain and the associated rule set for a diagnostic agent.

At the highest level of abstraction, the traditional model, depicted in Figure 1, consists of four entities: *Diagnostic Agent, Alert, Person*, *Infection*, and *Diagnostic Action*. Both *Diagnostic Action* and *Infection* are temporal and therefore contain attributes to indicate the applicable time span. The model indicates a *Person* may be further specialized as a *Doctor or a Patient and* may optionally have an (*has-a* association) *Infection,* which may be further specialized as a *Viral Infection* or a *Bacterial Infection*. Diagnostic actions are '*performed on'* a *Patient* and *'performed-by'* a *Doctor.* Two specializations of *Diagnostic Action* are defined: *Body Temp Measurement* and *Sore Muscle Check. Body Temp Measurement* is further specialized into *Oral Temp Measurement* and *Aural Temp Measurement*.

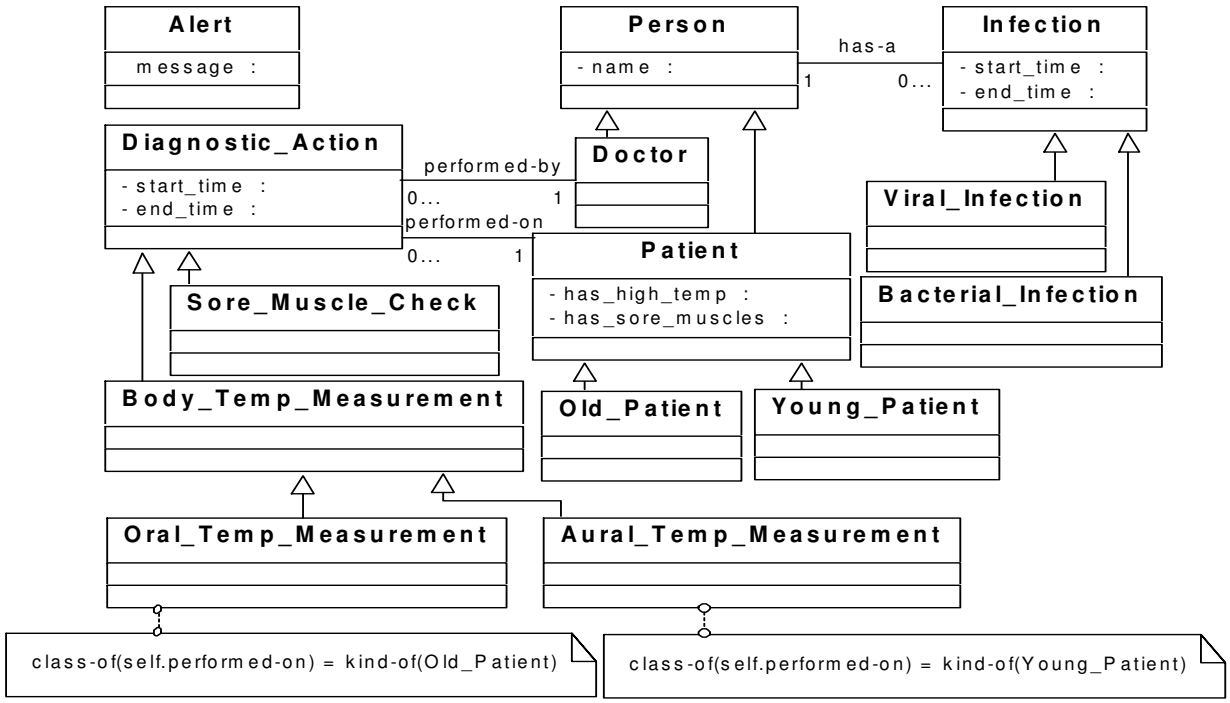

Figure 1: Traditional Model

The *Patient* class has Boolean attributes '*has high temp*' and '*has sore muscles'* to record the results of diagnostic actions. The *Patient* class is further specialized into *Young Patient* and *Old Patient*. The model also defines two constraints on diagnostic actions. *Oral Temp Measurement* may only be '*performed on'* an old patient, while *Aural Temp Measurement* may only be '*performed on'* a young patient.

The *Alert* class is provided so that the diagnostic agent can communicate recommendations and diagnosis with the user or other agents in the context of a larger system. Note that the model does not capture the associated diagnostic agent rules in any manner. These are specified in a declarative manner using condition action pairs as listed in Table 1. In all cases, the agent creates an Alert with the indicated message. The rule conditions specify patterns of linked objects specified in terms of the model class names and employ a priority to control the order in which triggered actions will be invoked.

The diagnostic agent is targeted to support infection diagnoses and is initially triggered when a person is known to have an undiagnosed infection by rules: 3, 4, and 5, which use instances of the *Alert* class to recommend appropriate diagnostic actions to '*perform on'* the associated *Patient*. In terms of the model, an undiagnosed infection is indicated by the association of an object that is a kind of *Patient* (instance of *Patient* or a subclass of *Patient*) to an instance of class *Infection* (not *Viral Infection* or *Bacterial Infection*). Rules 1 and 2 operate at lower priority than rules 3, 4, and 5 and use alerts to indicate a specific diagnosis.

The concrete nature of traditional models lends to their understandability, but detracts from their reusability in other domains. Reusability extends to the agent rules as well because they are directly dependent on the model. They also result in efficient implementations of agent behavior as modern languages natively support the classification hierarchies of traditional models upon which a large percentage of agent logic is typically based.

Table 1: Traditional Model Diagnostic Agent Rules

	Condition	Diagnostic Agent Alert Message	Priority
1	A kind of *Person* has sore muscles	*Person.name may have a viral infection*	1
2	A kind of *Person has high temperature*	*Person.name may have a bacterial infection*	1
3	A kind of *Old Person has-a Infection*	Perform oral temperature measurement on Person.name	2
4	A kind of *Young Person has-a Infection*	Perform aural temperature measurement on Person.name	2
5	A kind of *Person* has-a *Infection*	Perform sore muscle check on Person.name	3

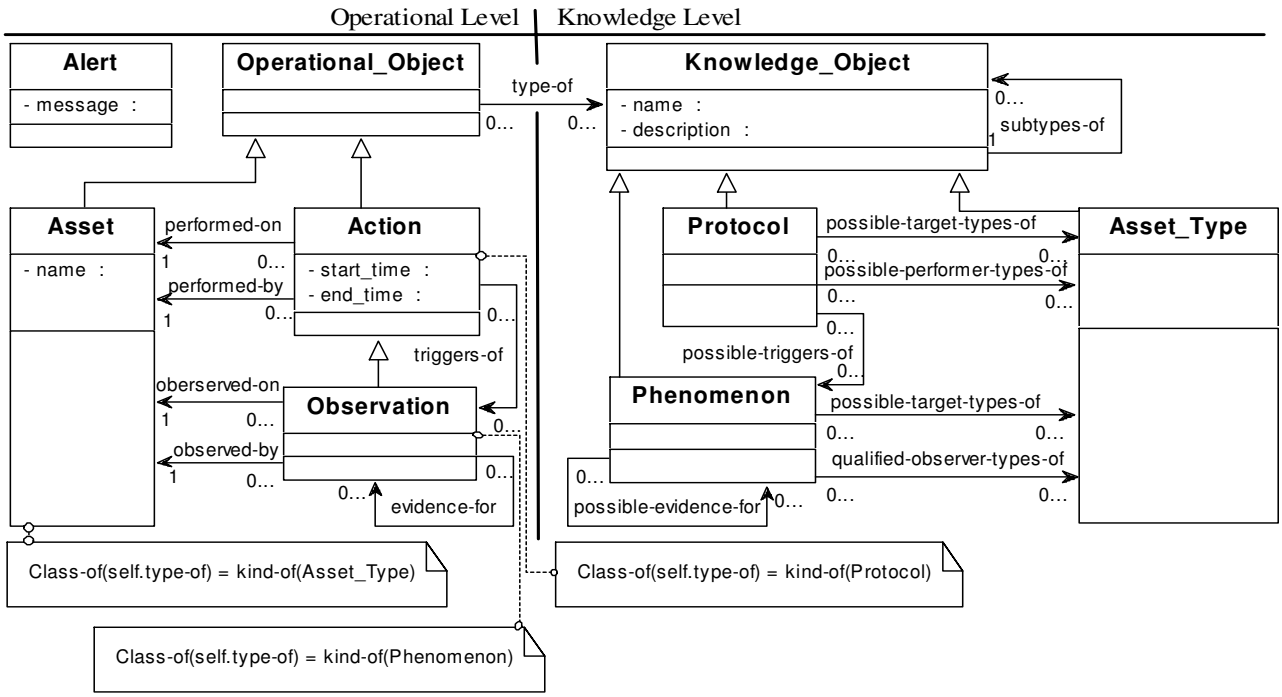

Figure 2: Split Knowledge Level Model

A primary drawback of traditional models is that the agent logic dependent classification hierarchies are not modifiable at runtime because modifications require recompilation of the class model. Another is that they do not readily support the commonly encountered concepts of Dynamic Classification and Multiple Classification.

A Split Knowledge Level Model

The shortcomings identified for the traditional model can be addressed by utilizing a more abstract model that is split into two levels: operational and knowledge. The split knowledge level model equivalent to the traditional model (Figure 1) is shown in Figure 2. In this model, the logical classification of operational level instances is provided by associative mechanisms rather than the native classification mechanisms provided by the implementation language, which is employed only for the inheritance mechanisms it provides to gather up the attributes, associations, and behaviors of a particular class of object.

In the split knowledge level model, operational level objects derive from the *Operational Object* class and knowledge level objects derive from the *Knowledge Object* class. The Operational Level Alert class is an exception in this simple example, but would also derive from Operational Object in a more complex domain that requires logically typed alerts. *Operational Object* instances capture the day-to-day occurrences and the specifically identifiable entities (Asset instances) within the target domain. The logical interpretation of an *Operational Object* instance is provided by its linkages to *Knowledge Object* instances and its context is provided by linkages to other *Operational Object* instances.

The *Operational Object* classes: *Action*, *Asset*, and *Observation* are more general and therefore more broadly applicable across a variety of domains. Given the appropriate linkages to knowledge level instances, to provide specific logical classification, these classes can be respectively substituted for the traditional model classes: *Diagnostic Action*, *Person*, and *Infection.* This approach allows the precompiled split knowledge level model to be logically tailored to a particular domain using runtime instances that capture the specialized or unique concepts within it.

The subtypes association of the *Knowledge Object* class is key to the understanding of the knowledge level. This association provides the mechanism to link object instances together to form taxonomies that can be iterated over at runtime to provide a more flexible classification scheme than that provided by the traditional model. The taxonomies that substitute for the classification provided by the class hierarchy in the traditional model are shown in Figure 3. The text at the top of the object instance depictions shows the value of the name attribute followed by the class of the instance. The *Infection*, *Person*, and *Diagnostic Action* classification hierarchies from the traditional model can be readily seen in the structures of

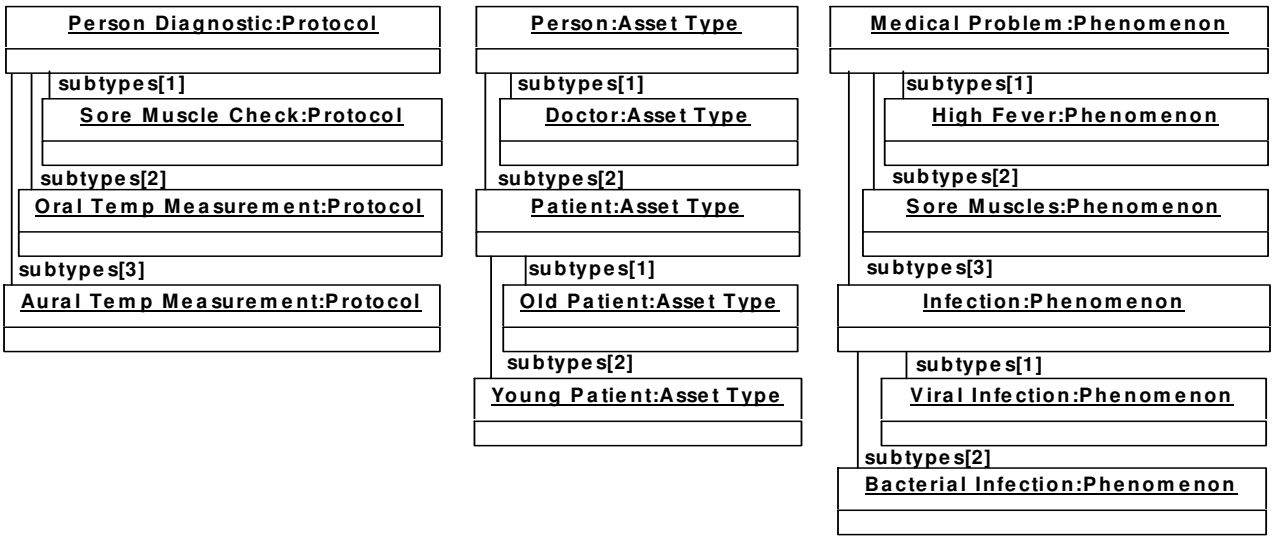

Figure 3: Knowledge Level Taxonomies

linked object instances of the respective *Protocol*, *Asset Type*, and *Phenomenon* classes.

Additionally the symptomatic phenomena '*High Fever*' and '*Sore Muscles*' are included in the *Phenomenon* taxonomy so that observations of them on '*Person'* Assets can be used to eliminate the need for the '*has sore muscles'* and '*has high fever'* attributes of class *Person* in the traditional model. This pattern of posting observations on phenomena to replace attributes of the *Asset* class partially eliminates the need for complex inheritance hierarchies that traditionally tie attributes to classes making a domain neutral statically compiled ontology a more feasible system design option.

In order to provide the same logical meaning as objects instantiated from the traditional model, an instance of an *Operational Object* class (*Action*, *Asset*, and *Observation*) must be associated with an instance of the corresponding *Knowledge Object* class (*Protocol*, *Asset Type*, and *Phenomenon*). In this manner, an instance of class *Person* in the traditional model is logically equivalent to an instance of class *Asset*, in the knowledge level model, linked to an instance of class *Asset Type* with an *name* attribute value of '*Person'* as shown in Figure 4.

While specific associations between: *Asset* and *Asset Type*, *Action* and *Protocol*, and *Observation* and *Phenomenon* could have been provided in the split knowledge level model a generalized '*type of*' association between the Operational Object and Knowledge Object classes was provided to pin down the conceptualization associated with the split at the top level. The generalization of these associations allows for generic implementations for '*type of*' and '*kind of*' methods that are applicable to all subtypes of the *Operation Object* class, but requires the addition of fixed constraints on the *Action*, *Asset*, and *Observation* classes, shown at the bottom of Figure 2.

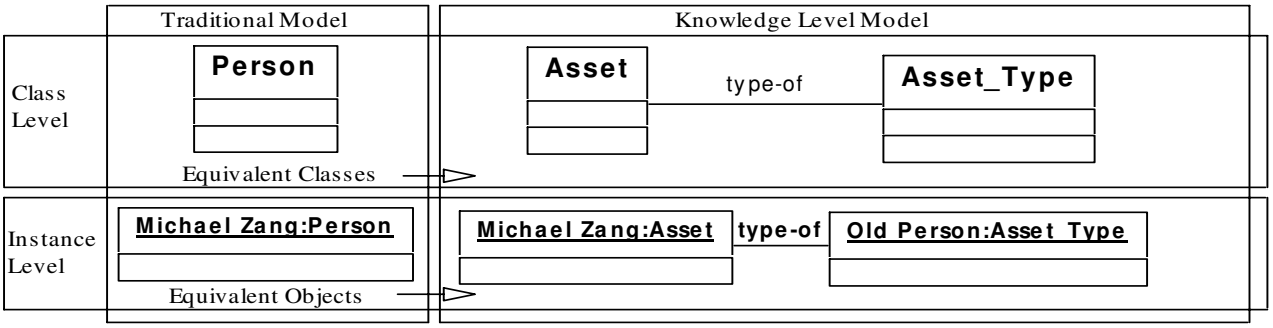

Figure 4: Knowledge Model Equivalence

Dynamic and Multiple Classification

In addition to providing support for extensibility at runtime, the knowledge level model also supports dynamic and multiple classifications, which are common in practice but difficult to implement using the traditional model. Dynamic classification refers to the ability of an object to change its classification at runtime while multiple classification refers to the ability of an object to belong to more than one classification.

For example, suppose a doctor gets sick and needs to be admitted to a hospital as a patient. With the knowledge level model, this situation can be represented by breaking the link between the representative *Asset* object and the *Asset Type* object with a *name* of *'Doctor'* and connecting it instead to the *Asset Type* object with *name* of '*Patient'*. With the traditional model this situation is much more difficult to deal with because the representative object and its classification are inseparable. The representative object of class *Doctor* must be destroyed and a new object of class *Patient* created, which results in a loss of identity. Although dynamic classification can preserve the individual identity of the *Asset* object as the logical classification dynamically switches from *'Doctor'* to '*Patient'*, reality is not being accurately modeled as the doctor is still a '*Doctor'* even when he is a '*Patient'*, only his role has changed. A better representational approach is to allow for multiple classifications.

Multiple classification allows a person to be both a doctor and a patient, and is easily represented in knowledge level model by associating both the '*Doctor'* and '*Patient' Asset Type* instances to the *Asset* instance representing the patient that is also a doctor. This is difficult to implement using the traditional model, which combines the concepts of inheritance and classification. In order to create objects that are classified as both a *Patient* and a *Doctor*, language provided multiple inheritance mechanisms must be used to create a new class *Doctor Patient* that inherits from both the *Doctor* class and the *Patient* class. While this in itself is messy, additional complications are incurred because the diagnostic agent rules (specified in Table 1) require that a patient be additionally classified as young or old; thereby, requiring additional usage of multiple inheritance to create classes *Young Doctor Patient* and *Old Doctor Patient*. This approach dilutes the clarity of the classification hierarchy and quickly becomes untenable in realistically scoped models.

Knowledge Level Rules

The knowledge level of the split model provides more than just taxonomies for logical classification. It contains associations for interlinking *Knowledge Object* instances, in a manner that records knowledge about the context of their usage. Unlike the purely rule-encoded knowledge employed with the traditional model, the knowledge recorded by interlinked *Knowledge Object* instances is in a form that is both dynamically extensible and conceptually accessible by system users.

Knowledge level associations eliminate the agent logic dependence on specific class names such as *Person*, *Young Person*, and *Old Person* seen in the rules from the traditional model (Table 1) by using set membership operations. These class names correlate to the named Knowledge Object instances in the split knowledge level model, which should not appear as hard coded strings within statically compiled agent rules as this prevents dynamically tailoring the model to different or changing domains at runtime.

The knowledge level defines two associations between the *Protocol* and *Asset Type* classes. The '*possible target types of*' association is used to indicate the asset types upon which a particular protocol is applicable. The '*possible performer types of*' association is used to indicate the asset types that may perform a particular protocol. The '*possible target types of*' association can be used to link the '*Oral Temp Measurement*' *Protocol* instance and the '*Old Person*' *Asset Type* instance to capture the fixed constraint from the tradition model that indicates oral temperature measurements should only be performed on old people. Similarly, the '*Aural Temp Measurement*' *Protocol* instance can be linked to the '*Young Person' Asset Type* instance to capture the other fixed constraint defined by the traditional model. The '*possible performer types of*' association can be used to link the '*Doctor' Asset Type* instance to the '*Person Diagnostic*' *Protocol* instance to capture the model constraint imposed by the '*performed by*' association between the *Doctor* and the *Diagnostic Action* classes in the traditional model.

The knowledge level also defines a 'possible triggers of' association to indicate the phenomenon that may trigger a particular type of action. The '*possible target types of*' and '*qualified observer types of* ' associations between the *Phenomenon* and *Asset Type* classes can be used to indicate the types of things a particular phenomenon may be observed on and the types of things, typically people or agent types, qualified to make an observation of the associated phenomenon. The '*possible evidence for*' self-association of the *Phenomenon* class is used to indicate phenomena that may infer the presence of specific phenomenon.

Table 2: Split Knowledge Level Model Diagnostic Agent Rules

	Condition	Diagnostic Agent Alert Message	Priority
1	*Observation* on an *Asset* of a '*type of*' *Phenomenon* in the 'possible evidence for' set of some parent Phenomenon	Asset.name may have parent Phenomenon	1
2	*Observation* on an *Asset* of a '*type of*' *Phenomenon* in the set defined by '*possible triggers of*' a *Protocol*	Perform *Protocol.name* on *Asset.name*	2

The equivalent of the Diagnostic Agent rules defined on the traditional model in Table 1 are shown in Table 2 for the split knowledge level model. The '*possible evident for'* self-association defined on *Phenomenon* allows a single domain independent rule (rule 1 in Table 2) to replace the two domain specific diagnosis rules developed using the traditional model (rules 1 and 2 in Table 1). The '*possible triggers of*' association between the *Protocol* and *Phenomenon* classes allows another domain independent rule (rule 2 in Table 2) to replace the three domain specific traditional model rules to recommend diagnostic actions (rules 3, 4, and 5 in Table 1).

By cross-linking the taxonomic concept hierarchies constructed using the subtypes association of *Knowledge Object* class, the essence of the rules developed for the traditional model has been moved into the form of instance data that can be readily modified or extended at runtime, while reducing the number of rules required. The rules that remain act as domain generic machinery for reasoning on the domain specific knowledge instance models. The domain specific knowledge instance models formed by interlinked *Knowledge Object* instances are created by subject matter experts or advanced users to tailor the statically compiled, domain independent model to support the specialized concepts with in the target system domain. By adding new knowledge level instances and linkages between them, which exist as data elements rather than code, an unlimited number of new concepts and rules can be added to the system at runtime.

While the split knowledge level model provides an infrastructure that supports modifications of agent behavior by allowing end-users to directly modify to the knowledge instance model that tailors the decision support system to their domain is not without issues, particularly for large knowledge bases. A common problem is determining if a piece of knowledge is actually new or just referenced by a different name in the current knowledge base. The next issue after identifying a piece of knowledge as new is identifying where in the complex interlinked hierarchical structure of knowledge level taxonomies should the new piece of knowledge be placed.

These issues can be partially alleviated through the employment of case base reasoning technology, which is well suited to identifying the level of similarity between *Knowledge Object* instances. An example of such is the Taxonomic Case-Based Reasoning System (TCRS) (Aha 2002)(Gupta 2001) that is currently being incorporated into some of the systems developed at CDM for the purpose of knowledge entry. TCRS is particularly well suited for this purpose as it employs taxonomically linked objects to tailor the question and answer dialogs associated with case retrieval to the level of expertise of the user. This makes it easy to correlate the internal representation of TCRS to that of a knowledge level based knowledge instance model.

Information and Knowledge

Commonly found within the professional literature related to the development of agent based decision support systems are characterizations of the progressively enriched concepts of: data, information, knowledge, and wisdom. These characterizations are typically abstract and conceptual in nature and are therefore difficult to relate to in practice. The structure provided by the split knowledge level model provides a concrete framework for exploring these concepts and concretely defining them within the context of a particular system implementation.

In a split knowledge level model based decision support system, data can be thought of as fragmentary bits of semi structured attribute value pairs resident in a variety of external sources. External data in which the decision support agents are interested is picked up by the external data broker for the system, which translates it into the operational level model of the system. Part of the translation process is concerned with putting the data into the proper context by linking it to the appropriate *Operational Object* instances currently residing in the operational level of the model. The data once put into context is raised to the level of information. The information content of the operational level is then periodically processed using statistical or other machine learning approaches to refine the linkages between *Knowledge Object* instances which are typically probabilistic in nature; thereby, refining the representation of knowledge within the domain. The generic domain independent rules, which determine how the system applies knowledge within the domain specific knowledge instance model, correspond nicely with the concept of wisdom.

The structure and elements of the split knowledge level model provide a rich environment for discerning the nature of and distinctions between information and

knowledge. The rest of this section describes some of the observations that can be made about information and knowledge by examining split knowledge level model implementations.

Information is a simplified reflection of knowledge. One can see the close parallels between the information and knowledge, as the structures are nearly mirror images of each other. However, one must remember that the instance model, not the class model of the knowledge level, acts as the meta-level for the operational level information, which for any realistically scoped system will be substantially more complex than the information model defined by the operational level classes.

Knowledge is derived from information and new information refines knowledge. As operational observations are linked together into an evidentiary chain using the '*evidence for'* association, the associated *Phenomenon* instances can be checked to see that they are similarly linked by the '*possible evidence for'* association. Knowledge level associations often take the form of probabilistic links whose values can also be adjusted based on this new information. In this manner, knowledge can be learned by observing operational information.

Information is dependent on knowledge for meaning. An *Action* or *Observation* is meaningless without the associated *Protocol* or *Phenomenon*. One could imply that something was done in the case of an *Action* with no knowledge level type but what was done could not be implied. Similarly, one could imply that something was seen, measured, or inferred in the case of an *Observation* with no knowledge level type, but what that something was could not be implied.

Knowledge exhibits more complex associations than information. This observation is exhibited in two ways. First, self associations in the knowledge level model must typically be implemented with reference objects or association lists as they may often associate the same object more than once, while the self associations in the operational level model can use the standard set implementation, as it is rarely the case that an information object needs to associate with itself. Second, associations between knowledge objects are typically probabilistic while those between information objects are direct.

Knowledge is more dependent on the domain than information. *Actions*, *Observations*, and *Assets* apply to just about any domain one can think of while the corresponding knowledge level entities *Protocol*, *Phenomenon*, and *Asset Type* can vary drastically from one domain to another.

Invalid knowledge should be temporally modified or destroyed. This only applies to temporal systems that allow one to go back in time. For example, diagnoses are made based on the knowledge of the times, which can change dramatically over time particularly in the area of medical diagnostics. This can be dealt with if all knowledge level objects and association classes have activation and deactivation dates, for example. In this manner, one can correctly judge past decisions based on the knowledge available at the time the decisions were made.

New knowledge allows a more precise specification of information. Consider again the medical diagnosis example. The Protocol 'Measure Body Temperature' could be extended with new knowledge that partitions 'Measure Body Temperature' into 'Measure Body Temperature Orally', and 'Measure Body Temperature Aurally' each of which has different margins for error. Using one of these new knowledge level *Protocols*, the example action on 'Measure Body Temperature' could be more precisely specified.

New information is easily identified whereas new knowledge is not. New information is always unique by definition. But consider for example a user of the model wishing to post an observation that his car engine is 'busted'; he does not find an existing 'busted' *Phenomenon* and therefore adds a new one. However, the Phenomenon 'broken' did exist. Are 'broken' and 'busted' the same? It is hard to say although a detailed study of the associated knowledge could help. For example, do they apply to the same set of *Asset Types*? This is a difficult issue to solve particularly if the associated system requires support for knowledge level extensions by the end users.

Information does not combine like knowledge. Information by definition is unique so that if two equally large collections of information are combined the result is simply twice the amount of information. Combining knowledge is much more complicated as it first involves identifying identical pieces of knowledge, particularly if dynamic extensions have been allowed, and then involves the combining of the probabilistic data associated with knowledge level associations.

Summarizing Conclusion

Split knowledge level models utilize an abstract, domain independent, statically compiled ontology divided into two distinct levels. The operational level provides classes to serve as templates for creating object instances that record the day-to-day events within the domain. The knowledge level provides classes to serve as templates for creating object instances to record domain specific

concepts and the knowledge of their application. Rather than using the language provided classification mechanisms operational level objects associate with knowledge level object to represent information related to their logical classification. This approach provides support for the powerful modeling concepts of dynamic and multiple classifications and allows for the development of generic statically compiled models that can be reused across multiple disparate domains.

The statically compiled knowledge level provides a control structure and generic rule activation mechanisms that system developers, subject matter experts, or advanced users may utilize to tailor the generic ontology to address the specialized or unique concepts within a particular system domain. Case base reasoning technology is particularly well suited to assist users in the process of placing new knowledge into the system as it can help determine if a piece of knowledge is actually new and if so it can help in determining the correct position within an existing taxonomy to place the new knowledge.

Ultimately the knowledge level is a structural layering pattern used in the specification of models for intelligent information systems. It is particularly well suited to the development of decision support systems as it provides a concrete framework with which to identify the data, information, knowledge, and wisdom associated with a particular system implementation. A well-designed model may be layered in other compatible dimensions as well and examples of this are provided in (Pohl 2000) and (Zang 2002).

A domain neutral, statically compiled model naturally leads to powerful domain neutral application components such as observation recorders, action schedulers, and taxonomy builders. Rather than hard coding such things as selection menu choices and graphical display layouts, applications query the ontological model at runtime to configure themselves appropriately for both the target domain and the current user. This sort of dynamic querying is very applicable to the highly optimized, statically compiled, procedural (albeit event driven and object-oriented) environments commonly employed in the development of graphical user interfaces.

Knowledge level models are not necessarily applicable to all software development projects. Although they reduces complexity by reducing both the number of classes and the number of rules, they increase complexity in other ways that make knowledge level models more difficult to understand, particularly for novice programmers and for experienced programmers new to a knowledge level based project. Knowledge level models are particularly applicable for use by development teams involved in the development of multiple (concurrent or over time) information systems that have focus on either intelligent agents or knowledge management.

References

Aha, David W. and Gupta, Kalyan Moy (2002), Causal Query Elaboration in Conversational Case-Based Reasoning; Proceedings of FLAIRS'02

Fowler, Martin (1997a); Analysis Patterns, Reusable Object Models; Addison Wesley Longman

Fowler, Martin and Kendall Scott (1997b); UML Distilled, Applying the Standard Object Modeling Language; Addison Wesley Longman

Gupta, Kalyan Moy (2001); Taxonomic Conversational Case-Based Reasoning; Proceedings of the fourth ICCBR, D. W. Aha & I. Watson (Eds.), Springer, Berlin, Germany, pp. 219-233

Hay, David C. (1996); Data Model Patterns, Conventions of Thought; Dorset House

Papurt, David M (1995); Inside the Object Model; Sigs Books

Pohl, Kym (2000); Perspective Filters as a Means for Interoperability Among Information-Centric Decision-Support Systems; Collaborative Agent Design Research Center, Cal Poly San Luis Obispo, Ca

Zang, Michael A (2002); Data, Information, and Knowledge in the Context of SILS; Proceedings ONR Workshop Series on Collaborative Decision-Support Systems; Office of Naval Research Logistics Program Office